Козлов Д.Д., Петухов А.А.

СБОР ИНФОРМАЦИИ О ПОВЕДЕНИИ ВЕБ-ПРИЛОЖЕНИЯ ПРИ ТЕСТИРОВАНИИ МЕТОДОМ "ЧЕРНОГО ЯЩИКА"

Лаборатория Вычислительных Комплексов факультета ВМиК МГУ
Москва
ddk@cs.msu.su, petand@lvk.cs.msu.su

Введение

На фоне неснижающихся темпов роста сети Интернет в сегменте конечных пользователей все большую популярность стали приобретать различные онлайн сервисы. Эти сервисы реализуются в виде веб-приложений, что позволяет использовать графический интерфейс, состоящий из стандартных элементов (HTML), единый транспортный протокол (HTTP), и не выдвигать особых требований к программному обеспечению клиентских рабочих станций. Вместе с тем, к онлайн сервисам предъявляются особые требования к качеству программной реализации: низкокачетсвенная реализация может не только испортить репутацию владельцев сервиса и снизить её конкурентоспособность, но и создать угрозы безопасности информации, обрабатываемой сервисом.

Одним из способов оценки качества программного обеспечения является тестирование. В зависимости от того, на какую составляющую качества [1] программного обеспечения направлено тестирование, выделяют следующие виды: функциональное тестирование, тестирование безопасности, нагрузочное тестирование, регрессионное тестирование и т.д.

Каждый из видов тестирования по-своему последовательно решает две задачи: задачу получения поведения приложения и задачу анализа поведения. В зависимости от возможностей, используемых для получения поведения приложения, выделяются следующие виды тестирования [2]: тестирование черного ящика, динамическое и статическое тестирование белого ящика. Если для получения поведения используются наблюдатель, использующий только внешние интерфейсы приложения, то говорят, что приложение расматривается как черный ящик. В этом случае под поведением понимается последовательность из пар (входные данные; выходные данные).

В настоящей статье рассматривается задача получения поведения веб-приложения при тестировании методом черного ящика. Для веб-приложений формулируется свойство «проигрываемости» поведения, анализируются существующие методы получения поведения

относительно указанного свойства и, наконец, предлагается метод для построения «проигрываемого» поведения.

Анализ задачи

Веб-приложение представляет собой набор отдельных программных модулей, работающих в составе веб-сервера. Каждый модуль получает HTTP-запрос в виде входных данных и возвращает HTTP-ответ в виде выходных. Для организации взаимодействия между модулями и сохранения информации между HTTP-запросами от одного пользователя используется разделяемое хранилище данных: файловая система сервера или СУБД, а также файловая система клиентских рабочих станций для хранения файлов cookie. Структурированный набор данных в таком хранилище называется *состоянием веб-приложения*.

Последовательность HTTP-запросов к веб-приложению от пользователя из-под одного контекста веб-клиента называется *сеансом работы* с веб-приложением. Проигрывание сеанса является *успешным*, если в результате выполнения всех запросов, его составляющих, веб-приложение ни разу не вернуло сообщение об ошибке. В противном случае проигрывание считается *неуспешным*. В случае детерминированных веб-приложений успешность проигрывания сеанса однозначно определяется текущим состоянием веб-приложения.

Сеансы могут изменять состояние веб-приложения. Так, успешное проигрывание сеанса может перевести веб-приложение в состояние, в котором проигрывание некоторых сеансов будет неуспешным. Аналогично, проигрывание сеанса может перевести веб-приложение состояние, в котором проигрывание некоторых сеансов будет успешным. Таким образом, можно говорить о зависимостях между сеансами по данным, которые составляют состояние веб-приложения.

Задача получения поведения веб-приложения при тестировании методом черного ящика заключается в получении упорядоченной последовательности сеансов работы с веб-приложением и такого состояния, что последовательное проигрывание всех сеансов из этого состояния является успешным. Опционально к этому требованию может добаляться требования обеспечения сеансами заданного покрытия.

Существующие решения

Поведение веб-приложения при работе с ним, как с черным ящиком, можно получить либо в результате непосредственного выполнения различных сценариев его использования, либо в результате журналирования сеансов работы с приложением других пользователей (например, настроив веб-сервер соответствующим образом). В первом случае возможны варианты автоматического и полуавтоматического (с участием оператора) взаимодействия с веб-приложением.

Автоматическое построение поведения дает наименее полные результаты. Инструментальное средство без дополнительной спецификации не знает о зависимостях сценариев использования веб-приложения по данным через общее хранилище. Поэтому, веб-приложение рассматривается ими не как автомат, а как граф. Задача построения поведения сводится к посещению всех ресурсов веб-приложения, доступных в данном состоянии.

Наиболее распространенный способ получения поведения веб-приложения в настоящее время – это запись сеансов работы с приложением его пользователей. В работах [4, 5, 6] описаны проблемы разбиения последовательности запросов, наблюдаемых на веб-сервере, на сеансы работы с веб-приложением отдельных пользователей, а также проблемы восстановления зависимостей по данным между этими сеансами. На настоящий момент указанные проблемы не решены в полном объеме [7].

В настоящей статье рассматривается вариант полуавтоматического построения поведения веб-приложения, так как он предоставляет наибольший контроль над рассматриваемым процессом.

Предлагаемое решение

Для решения поставленной задачи предлагается декомпозировать поведение веб-приложения на отдельные функциональные блоки, записать сеансы работы для каждого функционального блока, построить дерево зависимостей между функциональными блоками и на его основе построить последовательность сеансов для проигрывания. Далее предложенный метод описывается подробнее.

Совокупность состояния веб-приложения и сеанса работы с веб-приложением, который реализует определенный сценарий его использования, называется функциональным блоком поведения веб-приложения. Выполнение функционального блока является успешным, если проигрывание составляющего его сеанса является успешным. В детерминированных веб-приложениях функциональный блок $FB = (State, Session)$ является успешным, если проигрывание сеанса Session осуществляется из состояния State.

Функциональный блок FB_2 зависит от FB_1, если для успешного выполнения FT_2 требуется предварительное успешное выполнение FB_1. Примером таких зависимых блоков являются блоки, реализующие создание пользователя и аутентификация пользователя.

Функциональные блоки FB_1 и FB_2 являются зависимыми если:

- (прямая зависимость) FB_1 зависит от FB_2 или FB_2 зависит от FB_1;

- (транзитивная зависимость) существуют функциональные блоки FB_i, FB_{i+1}, .., FB_{i+k}, такие, что одновременно FB_1 зависит от FB_i, FB_i зависит от FB_{i+1}, …, FB_{i+k} зависит от FB_2.

Зависимости между функциональными блоками поведения веб-приложения могут быть смоделированы с помощью ориентированного графа: вершинами графа являются сами функциональные блоки, а ориентированные дуги отражают прямые зависимости. В случае, если от одной вершины до другой вершины имеется путь, соответствующие функциональные блоки являются зависимыми, в противном случае – независимыми. Необходимо отдельно отметить, что граф зависимостей функциональных блоков не является абсолютным для веб-приложения, а определяется его текущим сотсоянием.

Такой граф зависимостей может быть использован для построения проигрываемой последовательности сеансов работы с веб-приложением. Для этого необходимо воспользоваться следующим алгоритмом:

1. Зафиксировать вершины с нулевым количеством исходящих дуг. Присвоить этим вершинам метку глубины Depth = 0.
2. Последовательно обойти все неразмеченные вершины графа. Для очередной неразмеченной вершины:
   a. если все вершины, связанные с данной исходящими дугами, уже размечены, выбрать максимальное значение среди их меток, увеличить его на единицу и присвоить данной вершине;
   b. если есть неразмеченные вершины, связанные с данной исходящими дугами, выполнить их разметку по правилам пункта a.
3. Сериализовать полученный граф в последовательность вершин, начиная с минимальных меток и заканчивая максимальными.

Пример разметки графа зависимостей представлен на рисунке 1.

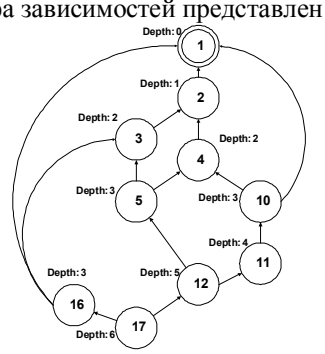

Рис. 1. Размеченный граф зависимостей

В результате применения алгоритма получится последовательность функциональных блоков поведения веб-приложения, которая может быть успешно проиграна из состояния веб-приложения, соответствующего первому функциональному блоку этой последовательности. Действительно, для проигрывания полученных блоков из начального состояния необходимо выполнить следующие шаги:

1. Сформировать множество пользователей веб-приложения, которые связаны с блоками построенной последовательности. Пронумеровать элементы этого множества от 1 до N.
2. Запустить N различных контекстов веб-клиента, пронумеровать их от 1 до N. Таким образом, каждому контексту веб-клиента будет поставлен в соответствие пользователь, связанный с некоторыми блоками рассматриваемой последовательности.
3. Последовательно выполнять блоки из последовательности в назначенных им контекстах веб-клиента.

Рассмотренный выше граф зависимостей функциональных блоков поведения веб-приложения и правило его сериализации используется для построения поведения веб-приложения. Для этого выполняется следующая последовательность шагов.

1. Фиксируется состояние веб-приложения. Для зафиксированного состояния строится граф зависимостей функциональных блоков поведения, затем размечается и сериализуется указанным выше способом.
2. Формируется и нумеруется множество пользователей, связанных с функциональными блоками полученной на предыдущем шаге последовательности.
3. Запускается соответствующее количество контекстов веб-клиента, каждый контекст ставится в соответствие уникальному пользователю.
4. Оператор последовательно выполняет сценарии использования веб-приложения, связанные с функциональными блоками обрабатываемой последовательности. Каждый сценарий использования выполняется из-под соответствующего контекста веб-клиента.
5. По мере выполнения каждый сценарий использования записывается в виде отдельного сеанса работы с веб-приложением либо на прокси-сервере, работающим между клиентом и сервером, либо самим веб-сервером при наличии соответствующих модулей и настроек.

Полученная последовательность сеансов работы с веб-приложением удовлетворяет критерию проигрываемости из состояния веб-приложения, зафиксированного на шаге 1.

Заключение

Рассмотренный в настоящей статье способ получения проигрываемого поведения веб-приложения имеет целый ряд полезных приложений на практике. Так, например, записанные один раз на этапе тестирования сеансы работы с веб-приложением, могут потом применяться для регрессионного тестирования после каждой значительной доработки программного обеспечения. Кроме того, описанный способ может применяться для тестирования безопасности веб-приложения, а именно его системы разграничения доступа. Действительно, возможность выполнения некоторого функционального блока без предварительного выполнения блоков, от которых он зависит, свидетельствует о некорректной реализации системы разграничения доступа веб-приложения (например, добавление сообщения на форум без аутентификации). Наконец, описанный метод может быть использован для повышения покрытия автоматических сканеров безопасности веб-приложений, которые реализуют тестирование на проникновение [3].

1. International Organization for Standardization. Software Engineering – Product Quality – Part 1: Quality Model. ISO, Geneva, Switzerland, 2001. ISO/IEC 9126-1:2001(E).
2. Patton, R. Software Testing, Second Edition, Sams, 2005, ISBN 0672327988.
3. J. Melbourne, D. Jorm, Penetration Testing for Web Applications, SecurityFocus, 2003.
4. Sprenkle, S., Gibson, E., Sampath, S., and Pollock, L. A case study of automatically creating test suites from web application field data. In Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications (Portland, Maine, July 17 - 17, 2006). TAV-WEB '06. ACM, New York, NY, 1-9, 2006.
5. Elbaum, S., Karre, S., and Rothermel, G. Improving web application testing with user session data. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 49-59, 2003.
6. Karre, S. Leveraging User-Session Data to Support Web Application Testing. IEEE Trans. Softw. Eng. 31, 3 (Mar. 2005), 187-202, 2005.
7. Sprenkle, S., Gibson, E., Sampath, S., and Pollock, L. Automated replay and failure detection for web applications. In Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering (Long Beach, CA, USA, November 07 - 11, 2005). ASE '05. ACM, New York, NY, 253-262, 2005.